The mystery of the writing that isn't on the wall: differences in public representations in traditional and agile software development

Conference or Workshop Item

The Mystery of the Writing That Isn't on the Wall: Differences in Public Representations in Traditional and Agile Software Development

Marian Petre
Centre for Research in Computing
The Open University
Milton Keynes, UK
m.petre@open.ac.uk

Helen Sharp
Centre for Research in Computing
The Open University
Milton Keynes, UK
h.c.sharp@open.ac.uk

Sallyann Freudenberg
Adfectio Ltd
UK
Sallyann.freudenberg@gmail.com

***Abstract*—This paper considers the use of public displays, such as whiteboards and papers pinned to walls, by different software development teams, based on evidence from a number of empirical studies. This paper outlines differences in use observed between traditional and agile teams and begins to identify the implications that they may have for software development.**

I. Public Displays in Software Development

This paper addresses one particular form of external representation (ER): public displays such as shared whiteboards or material posted on corridor walls. Cockburn [4] writes about these as "information radiators": displays posted where people can see them as they work or walk by, and which present relevant information, improving group communication with fewer interruptions. 'Good' information radiators are large and easily visible, can be understood at a glance, and change periodically.

Yet the nature and content of public displays in different software development environments can vary significantly. In traditional software development, informal representation features heavily [11], including whiteboards and paper displays on public walls. Many of the whiteboards and environmental displays observed in traditional settings reflect high-level structure (e.g., maps, structure diagrams, architecture diagrams) or planning (e.g., lists of tasks or requirements). In contrast, public displays in eXtreme Programming (XP) and agile software development focus on low- level components and project progress.

More broadly, XP takes a very different stance on representation in general from the long tradition of documentation and representation in traditional software development. For example, XP favours a verbal system metaphor over an external representation of the systems architecture; XP de-emphasises the usefulness of persistent formal external representations of the system; and so on [1].

Many researchers have portrayed the importance of external representations in design, where they are used both to support design reasoning and as a medium of communication among designers, both in design generally (e.g., [5]; [13]) and in software design in particular (e.g., [6]; [3]). Goldschmidt [8] describes a 'dialectic' between designer and sketched representation that contributes to the processes of design, evaluation, and reflection. Such research tends to focus on sketches used to explore or confirm design possibilities, whether individual or shared, on paper or on whiteboards.

This paper asks specifically: What roles do public representations (i.e., material displayed in shared spaces) actually play in supporting design? What might it mean if there are significant differences in what is represented in different settings, and might there be important implications, for example in terms of how a team's attention and communication are focused? Do the roles, include sharing designs and information, promoting communication and coordination within a design team, and developing designs, as the literature (e.g., [3]) might suggest?

Drawing on a number of empirical studies by the authors, this paper describes briefly the different contexts and considers the implications of those differences for software development. The paper is a reflection on empirical research, rather than a report on new research.

II. Observations About the Use of Public Displays

This section describes the public displays observed in agile and traditional software development contexts.

A. Agile Displays: Story Cards and the Wall

Sharp et al. [14] analyze in detail the activity of one agile team and conclude that the role of public displays is largely restricted to process issues such as progress-tracking, and that these artefacts lack detailed information about the application under development. This analysis also points out that, in a co-located team, artefacts are used in an information-rich environment supported by open and simple information flows. Sharp and Robinson [15, 16] analyze further teams and highlight the role of physical artefacts in co-operation and collaboration activities. They identify story cards and the Wall as key public displays with two main roles: enabling the capture of requirements and supporting the development process. Story cards represent user stories, the mechanism in agile development by which user requirements are captured. The Wall is a vertical surface on which the story cards are displayed publicly, in a codified structure that indicates their status (e.g., under development, ready for testing, etc.) and hence the progress of the project.

B. Traditional Displays: Whiteboards

Petre, over a number of empirical studies of professional software development [11, 12], observed that traditional

978-1-4673-1824-2/12/$31.00 © 2012 IEEE

Figure 1. (Agile) A photograph of the Wall, showing story cards arranged in an array that indicates status and priority.

teams employ a rich repertoire of informal external representations, offering differing perspectives on the software. Traditional teams' shared representations – on shared whiteboards and on papers pinned to walls – tend to be concerned with requirements, functionality, conceptual structure, and software architecture, with a certain amount of planning information (usually in the form of lists or annotations) juxtaposed. Whereas the agile 'Wall' is typically a single, focal representation, the shared displays in traditional teams are often more numerous, although there are typically key displays in places of high traffic. There are examples of 6-foot whiteboards propped next to desks or mounted in corridors and near coffee machines. Developers are observed standing around, glancing at them, gesturing toward them.

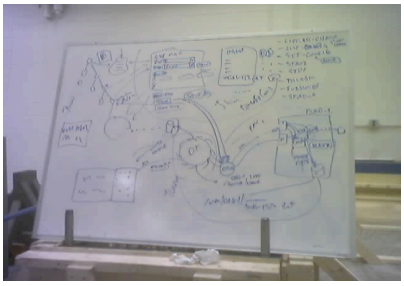

Figure 2. Fig. 2 (Traditional) A photograph of a whiteboard, showing structural and functional elements of a design.

III. The Implications of What Is/Isn't Displayed

Green and Petre [9] summarised a "Maxim of Information Representation": "Every notation highlights some kinds of information at the expense of obscuring other kinds." (p. 134) Not everything can be highlighted at once. If a representation highlights dataflow, then it may well obscure the control flow; if a representation highlights the conditions under which actions are to be taken, then it may obscure the sequential ordering of actions. If the public representation used by an agile team highlights the priority in which requirements are addressed, then it may obscure functionality; if the public representation of a traditional team highlights software structure, then it may obscure status or progress. Hence, the question here is: if the public displays of agile and traditional teams emphasise different information, where is the information maintained that is not displayed? Are there consequences on productivity of the choice of emphasis in the public representations?

A. It's in the Dialogues

One possibility is that the 'other information' resides in dialogues. Having public displays in frequented places serves to reinforce the information represented, promoting reference, awareness, and coordination. Information that is not represented explicitly is not reinforced across the team in this way. It falls to the social context to 'fill in the gaps': to share, renew, and articulate important information.

The public displays of agile teams emphasise the progress of the project as related to requirements. So where are functionality, structure and architecture captured? Beck [1] argues that a verbal system metaphor should be used instead of a representation of the system's architecture, and that the architecture should be instantiated through pair programmer dialogues and code generation, and that code should be largely 'self documenting'. But is this sufficient to support development and common understanding of a coherent, effective architecture? Other agile practices support sharing and the development of common ground between team members but (how) do these work in practice? Do all the members of an agile team maintain a consistent, coordinated view of the system architecture, or do components evolve independently, or do their dialogues promote 'sufficient emergent coordination' as it evolves?

The public displays of traditional teams emphasise functionality and structure. So where is the progress of the project reflected? Although progress information is not prioritised, and perhaps not monitored continually, there is some evidence that it is nevertheless valued. For example, one of the 12 criteria in the Joel Test [17], a 'straw test' to gauge the quality of a computer software team, is "Do you have an up-to-date schedule?" Do members of traditional teams maintain an awareness of the project priorities and progress, or concentrate on their own targets, perhaps proceeding in a manner that is not strategic to the project?

Both opportunities and issues are raised by relying on dialogues to maintain key project information. Doing so requires that dialogue actually happens; in agile dialogue is compulsory, whereas in traditional development it is a matter of good practice. Maintaining information (largely) through dialogue subjects the information to continual renewal: on one hand, the process of articulation and re-articulation may facilitate negotiation of shared understanding that develops along with the software artifact [2]; on the other, it may mean 'drift' in the understanding that poses a challenge for coordination between pairs or sub-teams. This is in contrast to dialogues around external representations, which can provide an 'anchor' for the negotiation of understanding and can assist in team coordination (to the extent to which the interpretation of the ERs is consistent across the team).

If the maxim is that representation is necessarily selective, then what accounts for the selections – and the implied priorities – associated with the different practices, and what are the trade-offs in terms of reasoning and outcomes? Is the implication that the other information is

less important for that development process or that it is more easily maintained through dialogue? Or that it does not require continual attention with the same support? Is the architecture of the software developed by agile teams so clearly established or stable that it does not need to be reinforced? Or is the social process enough to maintain coordination and address change? Similarly, is project progress something for which periodic attention is sufficient in traditional teams?

Although we've heard software developers speculate about the impact of 'the information that isn't on the whiteboard', our studies have not yet produced sufficient evidence to draw conclusions about consequences or tradeoffs.

B. It's in Someone's Notebook

Another possibility is that the other information *is* represented, but with less prominence, or in something other than a shared representation. Freudenberg [7] observed of agile teams that "there were … some indications of graphical representations of systems architectures in a number of project spaces, suggesting that these architectural diagrams were useful in group communications rather than when working on a specific development task". Sharp observed that agile teams refer to a planning phase in which an architecture is set out, but the representation of that architecture does not persist in public displays; it is enough that the team knows it exists 'somewhere'.

Similarly, process or progress information in traditional teams is often maintained by a project leader who keeps an overview, although that information may not be displayed publicly. Team members know what is expected of them, and they may monitor project progress beyond their own tasks via interactions and cues in the environment. Although not prioritised, some planning information may be included in public displays, typically as lists or as annotations to requirements or diagrams.

IV. SUMMARY

In general, the public displays of agile software development teams emphasise project progress and requirements at a low level of granularity. The public displays of traditional software development teams tend to emphasise higher-level requirements, functionality, conceptual structure, and software architecture. The differences in public displays between the two contexts raise a number of open questions that warrant further investigation:

- What does the dominant representation imply about priorities, or about the way the teams collaborate?
- Where does the other information reside (in the social interaction? elsewhere?), and is it maintained well enough to provide effective input into development?
- What are the implications of the differences? Would software development process be better supported by different public displays?

This example of provocative differences in practice in the two contexts highlights the need for comparative studies between the settings, using detailed analytic approaches to draw out factors that may affect reasoning, productivity, or quality. We plan to look more deeply and across other informal representations that support the development process and to seek to relate choices in representation to impact on efficacy.

REFERENCES

[1] Beck, K. 2000. Extreme Programming Explained: Embrace Change. Addison Wesley.

[2] Binti Abdullah, N.N., Sharp, H. and Honiden, S. 2010. Communications in context: a stimulus-response account of Agile team interactions. A. Sillitti, A. Martin,X. Wang, E. Whitworth (eds.) XP 2010, 166-171.

[3] Cherubini, M., Venolia, G., DeLine, R., and Ko, A.J. 2007. Let's go to the whiteboard: how and why software developers use drawings. CHI, 557-566.

[4] Cockburn, A. 2002. Agile Software Development, Addison-Wesley.

[5] Fish, J. and Scrivener, S. 1990. Amplifying the mind's eye: sketching and visual cognition. *Leonardo*, **23** (1), 118-126.

[6] Flor, N.V. and Hutchins, E.L. 1991. Analysing distributed cognition in software teams: a case study of team programming during perfective software maintenance. J. Koenemann, T.G. Moher and S.P. Robertson (eds.) Empirical Studies of Programmers: Fourth Workshop. Ablex, 36–64.

[7] Bryant (Freudenberg), S. (2004) Double trouble: mixing quantitative and qualitative methods in the study of extreme programmers. IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC) Rome, Italy.

[8] Goldschmidt, G**.** 1991. The dialectics of sketching, *Creativity Research Journal*, **4** (2), 123–143.

[9] Green, T.R.G., and Petre, M. 1996. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*, **7** (2), 131 - 174.

[10] Norman, D. 1993. Cognition in the head and in the world. *Cognitive Science*, **17**, 1-6.

[11] Petre, M. 2009. Insights from expert software design practice. ESEC/FSE'09, 233-242.

[12] Petre, M. 2004. Team coordination through externalised mental imagery. *Int'l Jnl of Human-Computer Studies*, **61** (2), 205 - 218.

[13] Schön, D. 1988. Design rules, types and worlds. *Design Studies*, **9** (3), 181–190.

[14] Sharp, H., Robinson, H., Segal, J., Furniss, D. 2006. The role of story cards and the wall in XP teams: a distributed cognition perspective. Agile 2006, 65–75.

[15] Sharp, H.C., Robinson, H.M., 2008. Collaboration and co-ordination in mature eXtreme Programming teams. *International Journal of Human–Computer Studies*, **66**, 506–518.

[16] Sharp, H.C., Robinson, H.M., and Petre, M. 2008. The role of physical artefacts in agile software development: two complementary perspectives. *Interacting with Computers*, **21** (1-2), 108-116.

[17] Spolsky, J. 2000. The Joel Test: 12 steps to better code. Joel on Software blog. http://www.joelonsoftware.com/articles/fog0000000043.html